THE X10ABOT MODULAR, EXTENSIBLE, ROBOTICS PLATFORM

A Thesis
Submitted in Partial Fulfillment of the Requirement for the
Degree of Master of Philosophy in Computer Science

of
The University of the West Indies

by
Rohan Anthony Smith
2013

Department of Computing
Faculty of Science and Technology
Mona Campus

ABSTRACT

The X10ABOT Modular, Extensible, Robotics Platform

Rohan Anthony Smith.

We present the architecture and deign of a general purpose robotics development kit, called X10ABOT, which aims to facilitate the rapid development of robotic solutions to a wide variety of problems. Robotics kits of this sort are usually aimed at casual hobbyists and children, and as a result, usually have severe limitations in the number of sensors and actuators that they can manage. In contrast, X10ABOT is: modular (new functionality is added through standard daughter boards), scalable (up to 16 sensors and actuators on each of up to 112 hardware modules), and extensible (module software can be upgraded to support new types of sensors and actuators). We have also designed a generic port, capable of handling several types of sensors; and we have done similarly for the actuators. We demonstrate the application of the platform to the development of a general purpose robotics kit that will show how the system would allow its users to readily add extra sensors and actuators to a project, with minimal configuration, and relatively little impact on the pre-existing user code.

Keywords: Robotics, microcontroller, firmware, architecture, simulation, real-time, mechatronics, automation, I2C, LEGO, Mindstorms, The X10ABOT Modular, Extensible, Robotics Platform, robotics framework; modular architecture, middleware.

ACKNOWLEDGEMENTS

I would like to acknowledge the support of my close family friends who served as the motivating factor that propelled me to finish.

I would like to offer my sincere thanks to my supervisor Dr. Daniel Coore, who has provided me with endless guidance and support. He has been an inspiration in my field and has provided me with insight and vast knowledge to help in the successful completion of this thesis.

Thanks to the Department of Computing, for providing me with the opportunity to further my studies and financial support through the Departmental Award.

To the members of my research group, I owe many thanks for the drive and motivation in completing my studies. The weekly meetings are an essential part of providing the momentum to complete.

Finally, this project could not have been complete without the strength and knowledge endowed upon me by God.

DEDICATION

This thesis is dedicated to my beloved wife and family. They have been there for me, even when I have all but given up. Thanks for keeping me going.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

1.1 Context

The `X10ABOT` robotics platform is a general purpose robotics kit that was designed to be a low cost, modular, scalable and extensible option for developing custom robotics projects. The system's intended audience and applications ranges from simple primary school level robots to more complex creations used at university level robotics competitions and research programs. The `X10ABOT` kit's focus was to improve upon current general robotics kit standards by providing:

- Support for potentially hundreds of sensors and actuators.
- A modular design with pluggable add-on peripheral boards.
- An extensible software and hardware architecture that easily supports new devices.
- A low overall cost while achieving all the above listed requirements.

Although many general-purpose robotics kits exist, many are either quite limited in the range of models that can be built with them, or demand a high price for their expressiveness. Many of the current robotics kits are designed with a limited number of available ports for attaching sensors and actuators. This is evident in the general purpose LEGO Mindstorms Robotics kit and most of the wheeled platforms like the Boe-bot and the Rug Warrior Pro **6frompaper** The underlying designs of these kits only partially support scaling, or not at all: one is forced to limit the features of their robotic designs to operate within this

limitation. Inadequate peripheral interfaces can curb creativity and innovativeness, resulting in unfulfilled designs, especially for robotics novices.

The `X10ABOT` robotics platform is a mix of both hardware and software. The hardware consists of two types of modules, a motherboard and daughterboard. These however can be implemented on a single piece of hardware with a single microprocessor or on their own physically separate hardware modules. For physically separate boards, they are connected via a bus that passes data and instructions between a single motherboard and one or more daughterboards. The daughterboards in turn processes these instruction and data and subsequently executes them using sensors and actuators connected to their respective ports.

There are specialized software components that are present on each hardware module. The motherboard hosts the user generated program code and the system software that processes it. It is also home for the middleware which is present on both the motherboard and daughterboard and is responsible for seamless cross-board communication. Finally there is the firmware which is embedded in the daughterboard, this is a set of semi-permanent software that was specialized to interpret and execute instructions for sensors and actuators, to and from the motherboard.

1.2 Thesis Summary

Outlined in this thesis are the details of how we progressed from our conceptual ideal product of a low-cost, modular, scalable, extensible robotics kit to transforming these ideals into realistic software and hardware components. These components were modeled and tested using the Arduino embedded development platform. We utilized one or more Arduinos to represent the motherboard daughterboard setup that we envisioned. We also took advantage of the the Arduino software framework which was used to develop the system

software, middleware and firmware, all user produced program code would also need to comply with the Arduino software framework.

The outcome of the use of these tools and strategies gave us a system that was able to easily control multiple sensors and actuators in a scalable and modular way while allowing for extensibility. The following code example will show how the `X10ABOT` platform manages the requirements we have outlined:

<code examples here>

Thesis Statement *It is possible to design an architecture for a general purpose robotics platform using the Arduino™ microcontroller and supporting software, open-source libraries and tools. It appears that by using a modular, distributed architecture for both hardware and software, it is possible is possible to create a low cost, scalable system with the ability to support hundreds of sensors and actuators. Finally we demonstrate that we can use this model to to support the system's extensibility through an API for hardware control and a liberal number of I/O pins and functionalities assigned to sensor and actuator ports.*

1.3 Structure of Thesis

Chapter 2 will explore the design goals of the `X10ABOT` platform, how the the same principles were being used by other robotics systems and how we applied them for our purpose. Chapter 3 describes the design and implementation of of the entire platform, the technologies used and how we integrated them to work seamlessly together. Chapter 4 will illustrate the results which will demonstrate the effectiveness of the design strategies we used. Chapter 5 explores the potential areas for future work and concludes the thesis.

CHAPTER 2 BACKGROUND AND MOTIVATION

Robotics development is a continually growing field with a promising future. the current sate of robotics can be compared to that of the PC industry in its embryonic years.**billgates** Many of the issues that the PC industry faced are redefining themselves in today's robotics arena. The main ones include high costs, a lack of common standards and generic platforms. Whenever it becomes necessary to develop a new robot for a particular task,it is often necessary to build it from scratch. This is because there are few general purpose platforms that meet all the needs of the typical robotics engineer, student and hobbyist**citation3** The "killer app" for the robotics field will be a platform that can address these issues and thereby create a broader adoption of robotics technology. As with the computer industry, one of the main markets that utilize robotics is education. Robotics is used to teach students of all ages and levels on topics ranging from science to social skills **citation11** General Purpose Robotic kits are one of the many 'edutainment' methods that are developed. These kits provide the user with the ability to build robots for a large range of applications.

An increased interest in robotics and robotics development has fueled the search for a robotics platform that will be useful for schools and the typical hobbyist. There has been research into finding a suitable platform that is flexible, inexpensive, extensible and capable of handling projects of varying complexities **suitable**; **bot-mate** Popular kits exist that meet some of these characteristics. For instance, the Lego Mindstorms™robotics kit is one of the most popular robotics platforms that can achieve many of these requirements. It however achieves a few of them through unconventional 'hacking' of the platform, or by purchasing third party equipment. This is not always easy to accomplish for

beginner developers. This project will outline the means by which the X10ABOT robotics platform provides for these specifications seamlessly through the features built into its design.

The X10ABOT robotics platform was designed from the ground up by integrating useful strategies found in other general purpose and custom robotics kits. This along with other innovations form the X10ABOT robotics platform's architecture. Innovations in both hardware and software have created a powerful system ready to build any robot for a school project or hobbyist invention. This was accomplished by focusing on achieving the combined goals of modularity, extensibility and scalability while maintaining a relatively inexpensive cost.

The motivating factors for this project extended from the need for involving robotics in education. The inclusion robotics in a school curriculum has been shown to improve performance in subjects of a technical or scientific nature **rbot** Robotics development allows for students to apply practical techniques to complete tasks and to solve scientific problems. The result or response from their work are immediately observable through the physical response of the robot. In addition, robotics present a 'fun factor' that attracts young people. To adequately utilize robotics as a viable educational platform, certain hurdles that were being faced had to be overcome. This lead to the creation of the X10ABOT robotics platform. This project was aimed at creating a platform that could adequately provide a full featured robotics framework for students, all at a price lower than the current market price for a full featured robotics kit. This will make possible the commoditization of robotics kits and subsequently allow for activities such as robotics competitions between schools. Other driving motivations for the project include the creation of a platform that would provide a flexible starting point for other robotics research projects. The kit should also be useful enough for hobbyist users who are interested in robotics and

automation, the X10ABOT robotics platform should be an attractive tool to utilize in their projects.

2.1 Similar Projects

We researched the structure of several robotics research projects and commercial robotics kits and identified a number of design objectives that were useful for a general purpose architecture. We aimed at creating a robotics framework that was able to support more sensors and actuators than most of the popular robotics kits, it had to support most of the common off-the-shelf sensors and actuators and yet be easy to use without limiting creativity.

We reviewed many types of robotics projects and came across a few interesting concepts. The following projects were encountered at varied intervals between design and completion, they both influenced and validated the design decision made in the X10ABOT project. We will review a few of them and an overview of their architecture.

The Tetrixx robotics project was developed in the late 1990's. It sought to overcome the issue of expandability being faced by its contemporaries such as MIT's 6.270 board and Handyboard projects. To overcome this limitation, the Tetrixx project employed specialized sensor and actuator expansion boards connected over a bus. The bus supported up to 64 expansion boards and allowed it to scale to far more peripherals than limited by the connectors from one single controller **tetrixx** This was one of the first systems which influenced the expansion board concept used in our project.

Another interesting robotics kit project was the LEGO Mindstorms NXT robotics development platform. This is a very popular robotics kit among schools and hobbyists. It has three actuator ports and four sensor ports on its

controller unit. Additional sensors and actuators can be added either by pairing with another controller or by purchasing third party multiplexing devices that allow multiple peripheral devices connected to one port. The port design is quiet extensible with numerous analogue and digital I/O pins and an I^2C bus to support even very complex devices. The Mindstorms kit has the distinct advantage of having a very active community of developers who have worked to create a large set of programming interfaces that can be used to write robotics applications for the platform. These include compilers for some of the most popular programming languages such as, C, Java, Python, Ruby, Ada, Lua and Matlab. The Mindstorms represented to us a very modern architecture and served as a standard by which we measured a lot of our innovations. The mindstorms however lacked the scalability we required, its other architectural attributes however were emulated in our design. It was also advantageous that most of the system was open sourced. We modeled our sensor and actuator ports after a similar design, we also took note of the software architecture and structured the `X10ABOT` to be able to facilitate a virtual machine type plugin that can interpret the Lego NXT bytecode. We started work on this but realized that the magnitude of effort was beyond the scope of this thesis.

We explored the architecture of the Hannibal Hexapod robotics project that had a commendable capability of managing the input and output signals for over one hundred (100) physical sensors and actuators. The Hexapod robot controller is based on the Subsumption Architecture and is fully distributed across eight(8) onboard computers: one each for the main controller, the robots body and each of its six legs. All these modules are connected via an I^2C serial communication bus. These components were the realization of the Hannibal Hexapod's design requirements of being scalable, modular, flexible, robust and adaptive. The Hannibal project greatly influenced the use of an I^2C peripheral bus as the central communication line on a distributed system of robotic components. This

seemed to have aided id the management of a large number of sensors and actuators. Since scalability was a desired attribute, we adopted this approach.

We also explored the architecture of the Rapid Robot Prototyping(R2P) project that had shared a lot of the design objectives that were outlined for the `X10ABOT` architecture. R2P's design goals were to be an inexpensive, open source, modular architecture for rapid prototyping of robotic applications **r2p** Similar to the `X10ABOT` , this architecture was aimed at students and hobbyists creating robotic applications on low powered microcontrollers. The architecture was built around a distributed system of modules communicating over a high speed serial data CAN bus. It emphasizes its plug and play capability by providing ports on each module to support daisy-chaining of multiple peripheral boards. On the software aspect, the R2P has embedded firmware on its peripheral modules to act as a Hardware Abstraction Layer(HAL) to encourage modularity. On the main controller, it implemented a Real Time Operating System(RTOS), a publish/subscribe middleware and a Virtual Machine that supports an embedded scripting language. The R2P system was encountered in the post-design stage of the `X10ABOT` project.It however validated a very large portion of the design of our architecture. The R2P system was very similar to `X10ABOT` , only differing in a few areas e.g. a CAN serial bus was used instead of I^2C serial bus. These examples influenced and validated some of our design decisions for the components of the `X10ABOT` architecture. We chose a combination of the best features of these technologies which we found to improve modularity, scalability and extensibility and combined them to construct our framework.

2.1.1 Modularity

Modularity in the design of a robotics kit architecture defines the capability to incrementally add fully independent sub-systems to a robotics project without

significantly modifying the existing configuration. According to Oraw and Tinder **mars** "The modularity of a robot is demonstrated by its expandable intelligence. Sensory modules that implement the robot's data protocol can be plugged into the system without reprogramming the original components". They also mention that the key to modularity is the use of peripheral modules. These were just a few of the design decisions made to satisfy our requirement of a modular architecture.

We chose to use of a distributed system as a part of the architecture. This allowed for the control of the overall system to be shared across multiple independently operating modules. These modules operate efficiently because there is very minimal usage of shared resources, this is because each module is usually self sufficient and simply passes information back and forth between itself and the main controller over the data bus.**show figure** In a centralized system which is dependent on one main processor, management of peripherals can become complex when handling a large number of modules **centralized figure** The controller's resources and connections are limitingly finite and there may be contention for these resources which may even prevent the inclusion of additional modules without the need to remove a previous module or somehow hack the system. Using a distributed system not only reduces complexity but it also allows for changes to be made across the entire architecture without affecting the existing setup**avcithesis**

A subsequent advantage of a modular architecture is software and hardware reuse, and rapid prototyping. Peripherals can be incrementally added to build more complex projects and subsequently reduce the overall cost of development**modcom** Different functions of the system can be delegated to independent and specialized modules. The aim was to simplify the control problem by decomposing the global task into local tasks for the robot's subsystems**paper[6]** This reduces the computational load on the main processor

by delegating the majority of the low level sensor and actuator processing to sub-modules.

Another common practice that many robotics architectures employ is the inclusion of peripheral sensor and actuator boards in their design. This can be seen in the Tetrixx project **tetrixx** and numerous other robots built around a distributed processing architecture. Peripheral boards are secondary computational units that are responsible for low level control operations. These peripheral boards interpret and execute data and control commands communicated between the main controller and the board's sensors and actuators. They are connected to a peripheral bus which will be able to uniquely identify each module along the communication line.

The peripheral bus that we selected supports up to 112 modules that can facilitate advanced robotics projects. Another requirement of the peripheral bus is high bandwidth that will be able to accommodate the possibility of a large number of active peripherals in a single session. We designed peripheral boards, named daughterboards that support sensors and actuators connected to the main controller over a fast, addressable serial peripheral bus. The daughterboards interact with the main controller board through a master-slave relationship. In our implementation, there exists just two types of components, the motherboard and daughterboards. Both type of modules have processors designated for their particular tasks, of which the motherboard may be required to be more powerful.

2.1.2 Scalability

Robotics kits that can support projects of varying complexities should have the ability to support as many sensors and actuators as may be required for a complex robotics project. A robotics project can range from robots that complete simple tasks that require one or two sensors and actuators to robots that manage

tens of sensors and actuators necessary to complete their task. As the complexity of a robotics project increases, generally, so does the number of peripherals required to complete the tasks. Projects that may solve reasonably technical tasks will sometimes have to improvise by finding creative ways to use available resources, however there may be times that the scale of the project requires a large number of sensors and actuators to adequately perform the task. Robots that are known for using a large number of sensors and actuators are usually ones that try to replicate animal or human ability. One such example is the Hannibal Hexapod Robot. Hannibal receives a tremendous amount of sensor information while continually controlling its almost 20 servo motors. The spider-like robot has over 60 sensors of different types **hannibal** Other types of robots that utilize many sensors and actuators include robotics arms, full humanoid robots, and animal replicas like MARS: the Modular Autonomous Robotics Snake **mars** In all of the above examples, the supporting architecture facilitated the inclusion and efficient management of a significant number of sensors and actuators.

We wanted to ensure that the cost of scaling was not prohibitive and that the developers would save on both time and money when implementing their large-scale robotics project. A distributed robotics architecture is key to decreasing the cost-to-scale factor. In a distributed system, the processing is shared between a central processing unit or a motherboard and numerous specialized modules. These modules are usually focused on doing simple tasks and therefore will not require complex hardware or devices to accomplish them. This can therefore allow them to to be inexpensive and simply connect to the motherboard to report the results of its computations and fetch data on request.

We needed a robust architecture that provided more than just an increase in connection points for sensors and actuators. It is critically important that there is adequate hardware and software support for the increase in additional devices. Proper power management is very important when managing tens to hundreds of

sensors and actuators. These devices can behave erroneously or simply fail to perform if they are not supplied with adequate power. It is also very important to place electrically noisy and high powered devices on separate power sources. This is especially true with motors and sensitive, low powered electronics should never share the same power source. This can electrically damage sensitive components of a controller and render the robot unusable. It is also necessary that the system be sized for its maximum power requirements and a capable power source selected. Under-powering a robot can also cause unpredictable results. Sometimes it may even be necessary to have multiple power sources on a robot that are adequately isolated. Our scalable architecture took these requirements into consideration.

Another requirement that may be easily overlooked is a structured and organized wiring and connection system. A robotics project can become increasingly difficult to develop if there is a web of wires and connections attached to the dozens of devices that may be present on a robot. A design that will mitigate this issue is the use of a distributed system supported by a main bus with an addressable serial data connection. If the architecture is designed with a single peripheral bus as its communication backbone which connects all devices, most of the wiring can be placed in a single organized package. The advantage of using an addressable serial bus is that these usually utilize very few connection lines. They also allow for new modules to be daisy chained onto each other. Any addition of a new set of sensors or actuators will simply equate to physically plugging a single set of wires wire from one module to another.

2.1.3 Extensibility

A truly extensible robotics system permits future addition and support of new sensors and actuators which might not have been included in the initial design.

Extensibility of a robotics kit provides developers with the freedom to define the hardware configuration with respect to the available types of sensors and actuators **rdk** A robotics kit may never be able to guarantee support for all types of sensors and actuators, but a truly extensible architecture will provides the means for developers to enable support for their particular hardware.

We used a middleware architecture to hide the low-level details of interface between the components that make up the `X10ABOT` architecture. This level of abstraction allowed for an interface between the main controller and the sensors and actuators which supported a high level definition of their operations. We saw this technique applied across numerous other platforms, this structural design is referred to as robotics middleware. “Robotics middleware provides functionality that is to be used and extended by the roboticists. Therefore, the tools provided by the middleware layer have to be capable of extending the robot’s functionality in order to be able to cope with its growing capabilities **advanced** ” There are a variety of ways to practically achieve extensibility through middleware but the general idea is to devise a method of allowing communication between the software on both sides of the middleware. This further abstracts the hardware interaction to the developer. Defining new hardware is now done through a configuration system and the middleware communicates information to and fro without exposing the communication protocol between the source and destination. This further hides any complex hardware setup and presents a flat structure even on a distributed system. Our system was designed to support as many types of sensors and actuators as possible. To achieve this, we had to find a common denominator for all sensors and actuators. Sensors and actuators have many requisite hardware technologies that need to be supported. We realized that these technologies were subsets or a combination of a finite set of operations. Technologies such as pulse width modulation, hardware interrupts, digital input/output and analog input are just

some of the basic ones used by most sensors and actuators. We determined that if we could define this complete set of hardware operations, we could control most if not all sensors and actuators. This would allow extensibility to currently existing and possibly new types of sensors and actuators.

2.1.4 Economics

An important requirement that will permit wider adoption of the robotics kit is ensuring that the kit's cost and its cost to scale are kept relatively low. The cost of electronics is is a decreasing function of time, however these prices are not always instantly obvious in the field of robotics. It is almost an accepted fact that robotics is an expensive field, the prices of robotics kits range from hundreds to thousands of U.S. dollars. This is a major point of restriction, robots are expensive and often beyond the resource of many persons and even universities and organizations who want to delve into the field of robotics **vr** In order to make a robotics kit that will be an attractive option for hobbyist roboticists and school programs even as low as the elementary school level, it has to be available at a competitive price point. A very popular platform for doing robotics is the Arduino development board which at the time of this publication ranged from U.S. $30 to $70. The Arduino development board is not specialized for robotics but has all the major tools necessary to build a functional robot if the developer is skilled in electronics. It is also one of the most popular development platforms at robotics competitions. The Arduino board also has a distinct advantage of being open-source. This gives the developer the advantage of having access to free software upgrades and longer hardware support. Being open-source also means that a skilled developer will be able to make modification to the system's code and even build their own hardware replica. These advantages can mean significant saving if a solution can be accomplished by the developer instead of purchasing an upgraded version or proprietary hardware.

CHAPTER 3 THE X10ABOT ARCHITECTURE, DESIGN AND IMPLEMENTATION

In this chapter we describe how we used the design strategies which were introduced in the background chapter. We extracted the most compelling features and combined them to crate a modular system with the following components which comprise the X10ABOT architecture:

1. The Motherboard
2. The Daughterboard
3. The Peripheral Bus
4. Input and Output Ports
5. The Motherboard Library
6. Daughterboard Firmware
7. The Middleware

The X10ABOT 's main architectural features that were decided on were modularity, scalability and extensibility. It was also important that we kept the cost of implementation relatively inexpensive. These attributes represented common design patterns observed over a number of robotics projects, either for custom robots or as a part of the architecture for general purpose robotics kits. We designed a system which comprises of components which realize one or more of the design requirements.

3.1 The Architecture

Based on the design requirements listed in the previous chapter, we created an architecture that captures all the previously listed features without making any significant compromise on any of those aspects. The `X10ABOT` architecture meets all the previously outlined requirements at both the hardware and software level. It is comprised of a motherboard, the main central controller and the head of a distributed system of peripheral sensor and actuator boards called daughterboards that carry out a standard set of computations synonymous to a thin client, all connected across the central peripheral bus. The motherboard coordinates and controls all operations on the entire system and all operations are either initiated or terminated at the motherboard. The motherboard hosts the main program for the robot application, the middleware and the extensible libraries to support various types of sensors and actuators. Each daughterboards is equipped with firmware to interpret and execute instructions given to it by the motherboard. It also provides multi-faceted sensor and actuator ports that can support a broad range of sensors and actuators.

3.1.1 The Hardware Architecture

The hardware aspect of the `X10ABOT` framework consists of three major components. These are the motherboard, daughterboard and the peripheral bus which connects them all. Below we will describe in detail how each component functions and how they relate to each other in accomplishing the goals of the architecture.

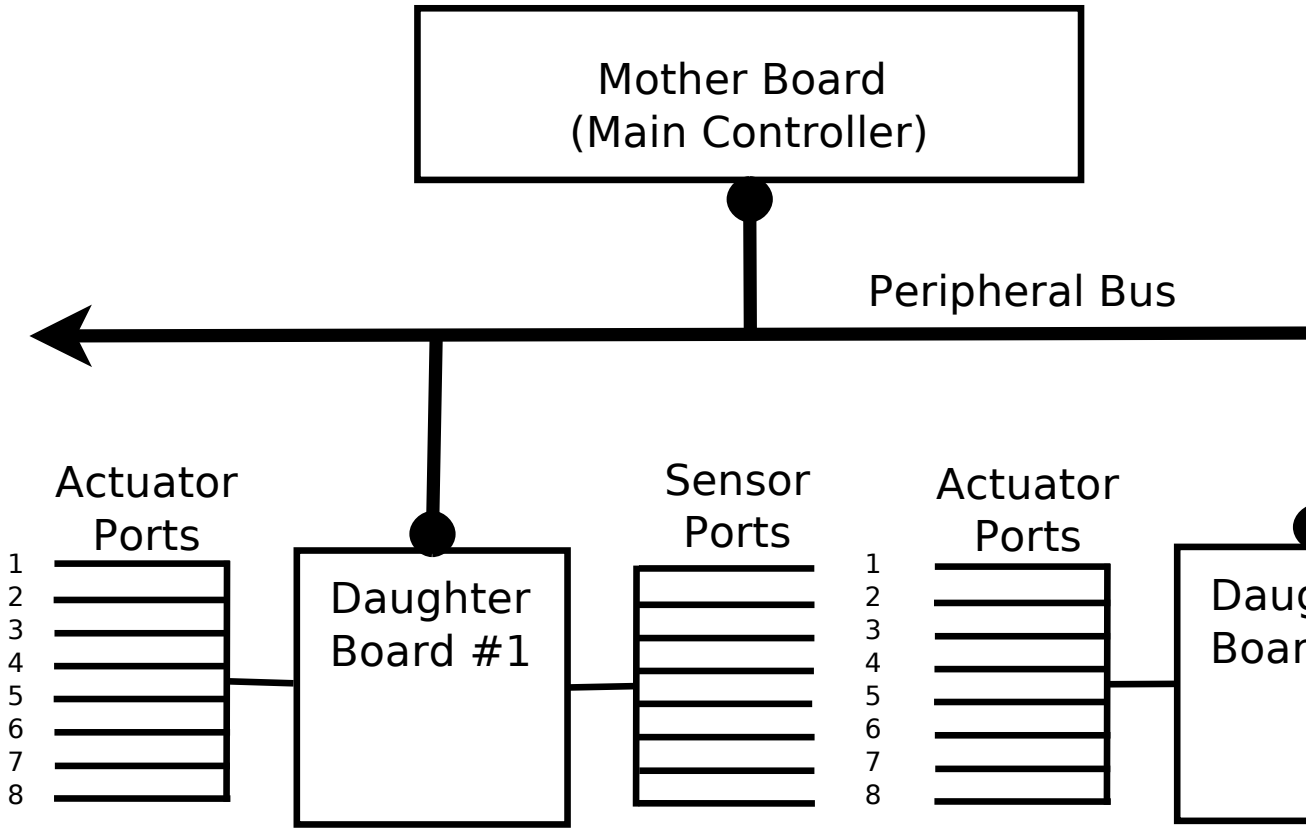

Figure 3.1. The components of the X10ABOT Architecture

3.1.1.1 The Motherboard

The motherboard is a physical component of the robotics platform and operates as the single central processing unit. it controls the flow of data and instructions across the entire X10ABOT architecture. All robot control instructions written by the robotics developer are hosted on the motherboard. The instruction logic is processed on the motherboard, however the actual sensor or actuator operations are dispatched to daughterboards as micro-code instructions. These microcode instructions are then subsequently interpreted and their operations executed by sensors and actuators across the entire system. As the main computational entity in the X10ABOT architecture, the motherboard is responsible for executing most of the decision logic, heavy computing tasks, actuating the output devices

and interpreting information from input devices based on its preprogrammed set of instructions. The motherboard acts as the head of the distributed system of peripheral boards. Sensors and actuators are hosted by daughterboards which are connected to the motherboard via the peripheral bus.

3.1.1.2 The Peripheral Bus

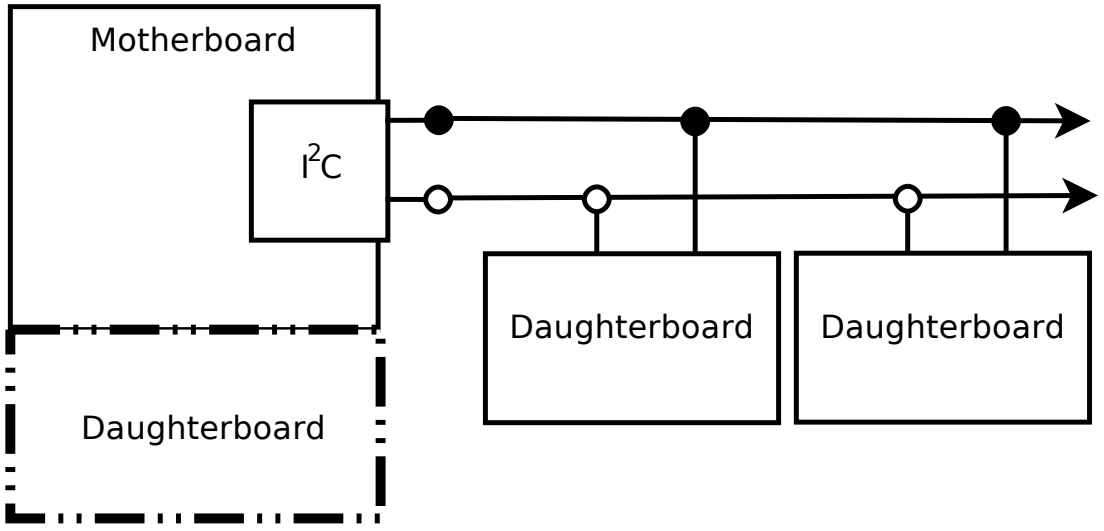

Figure 3.2. The serial communication peripheral I^2C bus

The Peripheral Bus is the central medium that connects the motherboard to one or more daughterboards and facilitates the communication of data and instructions between them. The peripheral bus is implemented using the I^2C serial communication protocol. I^2C is a multi-master, addressable slave, two wire bus serial protocol. It supports up to 112 uniquely addressable devices per bus (128 minus 16 reserved addresses), 8-bit oriented, bidirectional data transfers can be made at 100 Kbits/s in the Standard-mode, 400 Kbits/s in the Fast-mode, 1 Mbits/s in Fast-mode Plus, or up to 3.4 Mbits/s in the High-speed mode **indec** While operating in its standard-mode, the I2C bus can be as long as 9 - 12 feet without any significant noise interference **nonoise** This distance is quiet sufficient for a robotics kit where the peripherals are not usually more than 3 feet

away from the controller circuit. As such, I2C provides a low-cost solution for interconnecting large numbers of devices operating at relatively slow speeds, such as sensors or other external devices connected to a microcontroller **again** A number of robotics projects implement this method of data communication, including the popular NXT LEGO Mindstorms Kit, ISocRob, and the Hannibal hexapod autonomous robot **nuffadem**

All communication on the I^2C bus is initiated by the bus master or, as in our case, the motherboard. This is not suitable in all situations, e.g.: when a sensor detects an input that needs to be processed immediately. Even if all the daughterboards are periodically polled, the elapsed time may cause the sensor data to become invalid or ineffective. That is why we utilized the multi-master capability of a the I^2C peripheral bus to accomplish arbitration. I^2C inherently supports collision detection and bus arbitration, bus arbitration refers to a portion of the protocol that ensures that when multiple masters try to control the bus simultaneously, it allows only one master full control of the bus while queuing the prospective master without any corruption or dataloss**nxpi2c** The motherboard has primary control over the peripheral bus but periodically gives up the control to any daughterboard that wants to gain temporary bus-master status. The capabilities given to a daughterboard that has claimed bus-master status is limited. Daughterboard which claim bus-master status can only communicate with the motherboard, there is no daughterboard-to-daughterboard communication. The daughterboard cannot request data from the motherboard, it can only transmit its own data. The peripheral bus allows for easy connection of additional devices because they can be daisy-chained onto each other reducing the need for extensive and confusing wiring. Since the peripheral bus is an I^2C bus, it is compatible with "non-daughterboard" I^2C compatible devices. The framework was designed to accommodate sensors and actuators as specialized, single device daughterboards. This was also a method of ensuring maximal

extensibility.

There is one exception to the use of the I^2C bus as the main peripheral bus, this involves the use of an internal daughterboard. We will expound on this concept in the next section.

3.1.1.3 The Daughterboard

The daughterboard is the secondary computing device in the `X10ABOT` robotics architecture. It hosts all the sensors and actuators that carry out the tasks of each robotics project. Although the daughterboard is a second tier processing component in the X10ABOT architecture, its operation is not trivial. The daughterboard is given the major task of interpreting instructions sent from the motherboard over the peripheral bus and executes them with sensors and actuators. Sensors and actuators are connected to sensor and actuator ports respectively and respond to the assertions placed upon them by the daughterboard. Daughterboards carry out simple low level operations and won't require a significant amount of processing power. There are however a few basic features that every daughterboard must possess. All daughterboards must be able to communicate over an I^2C bus and do basic digital and analogue input and output. These features are very common in most microcontrollers and can be implemented without a significant cost or effort. As a member of the peripheral bus, each daughterboard must possess a unique address that the developer is aware of, it also should have knowledge of the motherboards address. All motherboards will have the same address since only one motherboard is allowed on the peripheral bus at a time. A daughterboard will need the motherboard's address whenever it has time critical data that needs to be sent to the motherboard. Using bus arbitration, the daughterboard tries to

temporarily claim the bus-master role in order to quickly pass its data to the motherboard, it would then immediately switch back to its slave role.

3.1.1.4 Internal Daughterboard

On a typical motherboard circuitry, the only external IO pins that are generally utilized are SCK and SCL that operate the I^2C protocol. This leaves a large number of unused IO pins that could have otherwise been used for some kind of operation. We then came up with method for utilizing these extra resources without breaking design of the architecture. We design a virtual, internal daughterboard, that uses the same microprocessor and physical board as the motherboard. This internal daughterboard is assigned with address number zero(0). All its data and instructions are communicated using direct parameter passing instead of across the peripheral bus. This particular exception is abstracted by the middleware 3.1.2.4 and is totally hidden from both the motherboard and the internal daughterboard.

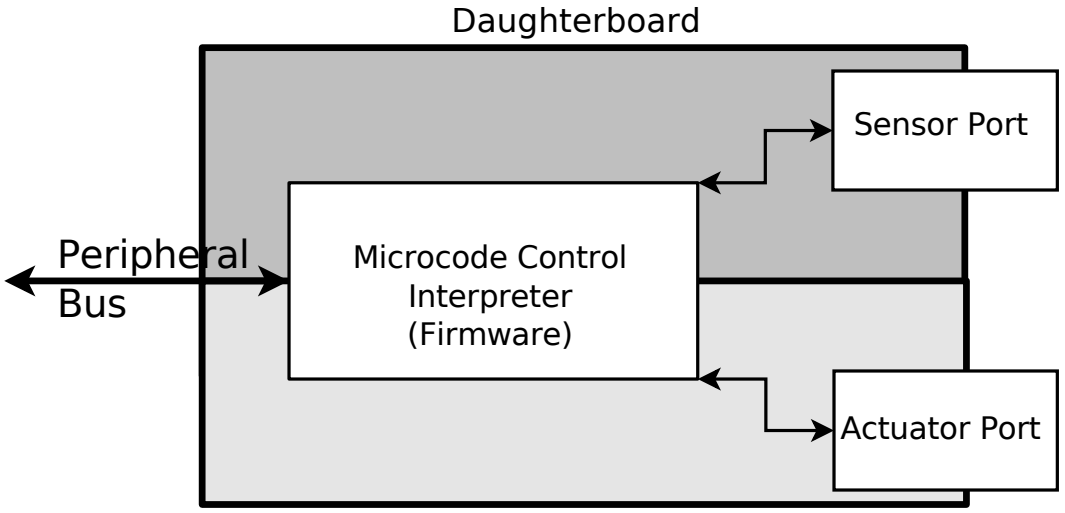

Figure 3.3. The daughterboard slave module

3.1.1.5 Sensor & Actuator Ports

All sensors and actuators connect to daughterboards to send and receive data through their respective sensor and actuator ports. Ports are specially designed interfaces that were made to facilitate all types of sensors and actuators. They were built with the most common technological requirements of popular sensors and actuators. The ports are made of up to six(6) pins connected to the daughterboard which should suffice for most operations a sensor or actuator may need to perform.

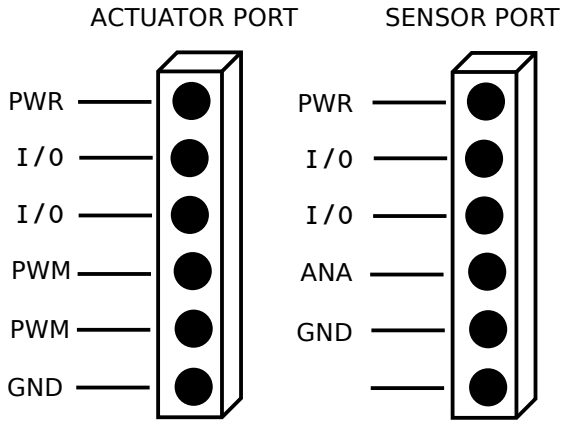

Figure 3.4. The sensor and the actuator ports.

A sensor port is comprised of five(5) pins: power and ground, an analogue input, and two digital I/O pins. The analogue input pin is able to read voltages from analogue sensors e.g. touch, light and sound sensors. The digital I/O pins can support active and passive digital sensors such as ultrasonic rangefinders, passive infrared(PIR) sensors and Radio-frequency identification(RFID) sensors. The power and ground pins provide electrically isolated power for the sensors (typically 5 volts, but in principle it could be different from the microcontroller's voltage requirements).

An actuator port is comprised of 6 pins: power and ground, two (2) Pulse Width Modulation (PWM) output pins and two (2) digital I/O pins. The PWM signals

allow support for servo motors as well as for adjusting power levels on output devices. The digital I/O pins are able to drive dc motor H-bridges or to be read as digital encoder signals. Like the sensor ports, the power supply for output ports is also electrically isolated from that of the microcontroller. This is particularly necessary when powering motors to prevent damage to the controller and other electronics.

A power bus runs through each daughterboard, connecting all the power pins of its ports. This power bus is meant to supply an alternate power source to the attached sensors and actuators. This alternate power source may or may not be suitable for powering the internal circuitry on the daughterboard, therefore, it is properly isolated. The power bus terminates as connectors on either side of the board This facilitates daisy chaining the power across the daughterboards, or if necessary, providing separate power supplies for individual modules. Indeed, the scalability of the `X10ABOT` architecture is more likely to be limited primarily by power considerations than by controlling software or the number of available ports.

3.1.2 The Software Architecture

The `X10ABOT` hardware is managed by software distributed across both the motherboard and daughterboard that is responsible for the efficient operation of the system. The software managing the architecture can be classified into four categories: firmware, middleware, device libraries and application software. The application software, libraries and middleware reside on the motherboard while the daughterboards hosts the firmware. Unlike the hardware architecture, we will be describing the software with direct relation to our particular implementation on the Arduino embedded system platform.

3.1.2.1 The Motherboard Library

The motherboard robotics library is a toolchain of functions written specifically for robotics applications. We defined and developed an extensible library of useful robotics operations for common devices and general sensors and actuators. They can be composed to control the behavior of a robot. Our particular implementation made use of the Arduino platform which supports the addition of third-party libraries. This allows for robotics developers to take advantage of the usability of the Arduino development platform while capitalizing on the usefulness of a robotics-specific set of operations implemented in the library. The library was designed not only to provide common robotics functionality to its users but to also make programming large robots with many sensors and actuators less of a hassle especially when taking advantage of the scalability and modularity of the entire platform.

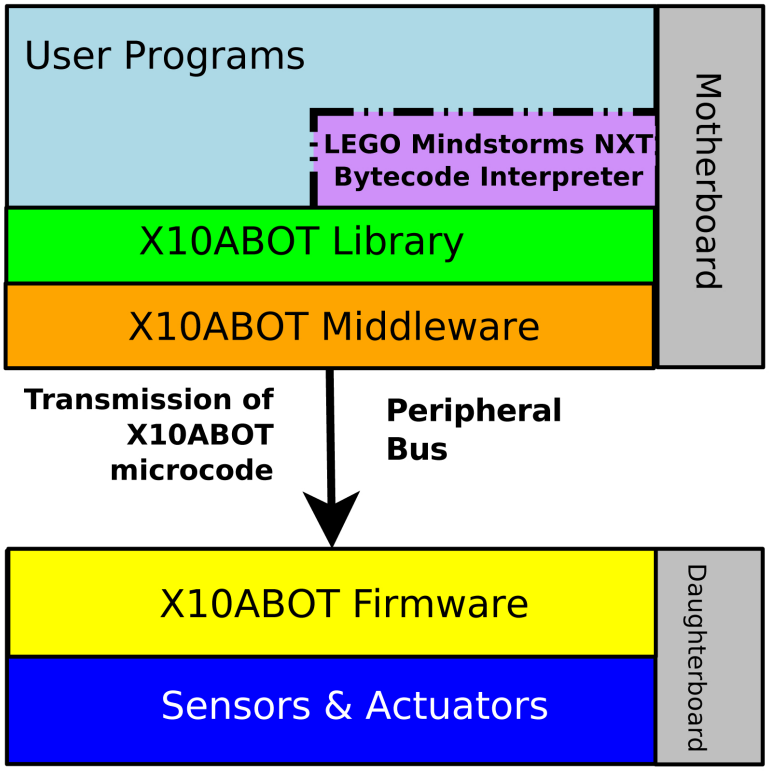

Figure 3.5. The X10ABOT software architecture

3.1.2.2 Application Software

Code written by users that takes advantage of the features of the X10ABOT architecture to complete a particular task is considered to be application software. The main purpose for creating a robotics framework is to allow the end user to have a gratifying experience when developing software for their robotics project. We wanted to provide the user with the tools that will allow them to build more interesting robotics projects, without the usual limitations of scale or complexity that may follow.

We used Arduino's Processing language as our choice for the implementation of the framework. It is know to be very easy to learn and use and also has a supporting IDE that allows the user to get up and started in a short period of time. Applications will be written in a C-like syntax and compiled to run on an

Arduino board. This is a familiar interface for many roboticists because Arduino is a very common robotics development platform.

3.1.2.3 The Daughterboard Firmware

The daughterboard firmware allows it to provide a standard interface from all daughterboards to their motherboard. All instructions to control or access sensors and actuators are communicated to it via the peripheral bus. These instructions are written as microcode abstractions of the hardware and need to be translated to actual assertions on the devices. Each abstracted command in the motherboard's `X10ABOT` Library is translated into one or more microcode hardware instructions. Microcode instructions are carried out sequentially by the daughterboards to manipulate the physical hardware components that are attached to their ports. The microcode instructions define a universal set of hardware operations. When these are logically put together, they can control virtually any typical robotics sensor or actuator. The microcode instructions specified for daughterboards, provide different means of interacting with each port (see Table 3.1). The firmware on the daughterboards will interpret these instructions and execute hardware procedures. These instructions are asserted to the standard sensor and actuator ports which are composed of pins, each with their specific purpose. Sensors and actuators are connected to their respective ports and respond to the assertions placed upon them by the daughterboard.

Category	**Input/Output**	**Data**
Digital I/O	Read/Write	HIGH/LOW
Analogue	Read	n/a
PWM	Write	Frequency and Duty Cycle
I^2C	Read/Write	Serial Data
RS-232	Read/Write	Serial Data
Watch	Rising/Falling Edge, Threshold	Threshold Value

Table 3.1: The complete set of hardware operations supported by the microcode interface to the daughterboards.

The microcode instructions are a universal set of hardware operations restricted to the types of sensors and actuators that are in common use (e.g. analogue, digital, serial, PWM). If a future sensor or actuator should use an incompatible operation, then a firmware upgrade which would include the new microcode instructions would have to be developed in order to interface with it (assuming that it could be made to be hardware compatible with the port). The microcode instructions are made to be very generic, ensuring compatibility with as many sensors and actuators as we could find at the time of design.

The motherboard implements a number of functions that allow the libraries to utilize microcode instruction in the development of device libraries. These instructions are passed to the daugterboard were they are interpreted and executed. Code example 1 shows a few of the microcode instructions used in the `X10ABOT` library and their related parameters.

Microcode Functions

```
/**
 * @param  byte  state  The state of the digital pin, HIGH(2), LOW(1), INPUT(0)
 * @param  byte  db_address The daughterboard address , 0 for motherboard
 * @param  byte  port_number The daughter board's port number connected to the device
 * @param  byte  pin_select Choose which of the I/O pins on the port to use A(0) or B(1)
 * @param  byte  pwm_select Choose which of the PWM pins on the port to use A(0) or B(1)
 * @param  byte  duty_cycle Specify the duty cycle as a number between 0 and 255
*/
  void digitalOut(byte state, byte db_address, byte port_number, byte pin_select);
  byte digitalIn(byte db_address, byte port_number, byte operation);
  void pwm(byte pwm_select, byte db_address, byte port_number, byte duty_cycle);
  int analog(byte db_address, byte port_number);
```

Listing 1: Example of the `X10ABOT` microcode instructions used in the Library.

For every event on the daughterboard that is to be triggered, a microcode instruction is sent from the motherboard. The format of the instructions are as

follows: The first four bits of the first byte define the instructions to be executed and the next four bits define the specific operation for that instruction. The third byte specifies the daughterboard address and the fourth byte has seven bits for the port number and one bit for the pin selection. Each transmitted instruction has a a sequence number that that is assigned to each microcode instruction in the sequence that it is executed. Some instructions require additional data to complete its operations. The microcode format provides a sequence of data bytes that can be of arbitrary size, it is used in operations where a lot of data maybe transmitted e.g RS-232. **tab:mc-structure** shows the structure of a typical microcode instruction.

Byte 1:	1111XXXX	FUNCTION BYTE
Byte 1:	XXXX1111	OPERAND BYTE
Byte 2:	11111111	D.B. SELECTION
Byte 3:	1111111X	PORT SELECTION
Byte 3:	XXXXXXX1	PIN SELECTION
Byte 4:	XXXXXXX1	FUNCTION SEQUENCE NUMBER
Byte 4|+n	11111111	DATA BYTES; n> 0

method named on

Table 3.2: Byte layout for transmitted microcode

Code example 3.1.2.4 shows how microcode instructions are used to compose a particular robotics instruction in the `X10ABOT` architecture. Here the method named **"on"** is defined for the Actuator class. The **power** paramenter is passed as an argument to the function. For negative and positive values of **power**, separate actions are taken. the same microcode operations are executed but with different parameters.

Robot Instructions

```
void Actuator::on(byte power){
  if (power>0)
  {
    actuator.pwm(A, _db, _port, 255*power/100);
    actuator.digitalOut(HI,_db,_port,A);
    actuator.digitalOut(LO,_db,_port,B);
  }
  else{
    actuator.pwm(A, _db, _port, 255*abs(power)/100);
    actuator.digitalOut(LO,_db,_port,A);
    actuator.digitalOut(HI,_db,_port,B);
  }
}
```

Listing 2: Example of the a robotic instruction which is comprised of microcode operations.

The purpose of the daughterboard is to interpret these high-level instructions from the motherboard and execute them on the respective sensor or actuator irrespective of the hardware specifics of the daughterboard. This ensures that high level language instructions sent from the motherboard remains platform independent.

3.1.2.4 Middleware

Bakken et al. **bakken2001middleware** defined middleware as follows: "a class of software technologies designed to help manage the complexity and heterogeneity inherent in distributed systems. It is defined as a layer of software above the operating system but below the application program that provides a common programming abstraction across a distributed system." The portion of the X10ABOT software architecture that performs the role of middleware does so

by providing the motherboard with a level of abstraction and seamless communication with the daughterboards. The middleware handles communication between devices without exposing the details of the protocol or its requirements and the details of the systems hardware on either end. In fact the middleware is used to determine what type of communication medium and protocol is appropriate and creates a standard platform independent interface for interaction. In other words, The middleware just simply presents a standard interface that communicates data and instructions between both the motherboard and daughterboard. In a review of popular robotics middleware, Kramer **Kramer2006** defines cites some major advantages which are applicable to the `X10ABOT` architecture. He states that it aids in software modularity, which is an intentional design objective. The middleware allows for pluggable libraries at the motherboard level which are ignorant of the actual hardware implementations. This accomplishes the platform independence, and portability that Kramer found to be a common attribute of most robotics middleware.

The aspect of the `X10ABOT` architecture that transports the microcode from the libraries of the motherboard to the interpreter of the daughterboard is considered to be our middleware. Each microcode instruction formats the byte string as described in Table 3.1.2.3. The microcode then performs one or more of a series of dispatch and request events between the motherboard and the daughterboard. The daughterboard also initiates data transfers as well for certain special cases. All these operations are managed by the middleware.

Robot Instructions

```
void X10ABOT_MB::digitalOut(byte state, byte db_address, byte port_number, byte pin_select){
  byte microcode[] = {FN_IO+state,db_address,((port_number-1)<<1)+pin_select,incr_instr_seq()};
  dispatch(microcode, sizeof(microcode));

}

byte X10ABOT_MB::digitalIn(byte db_address, byte port_number, byte pin_select){
  byte seq_num = incr_instr_seq();
  byte microcode[] = {FN_IO+IN,db_address,((port_number-1)<<1)+pin_select,seq_num};
  dispatch(microcode, sizeof(microcode));
  delay(50);
  return requestHandler(microcode, 2,seq_num);
}

void X10ABOT_MB::pwm(byte pwm_select, byte db_address, byte port_number, byte duty_cycle){
  byte microcode[] = {FN_PWM,db_address,((port_number-1)<<1)+pwm_select,incr_instr_seq(),duty_cycle};
  dispatch(microcode, sizeof(microcode));
}
```

Listing 3: Example of the a robotic instruction which is comprised of micro-code operations.

The dispatch and request events are used to

3.1.2.5 Extending the Library

To allow for easy extensibility and to support new hardware, the X10ABOT software architecture was designed to allow user extensible libraries. A library defines a set of microcode instructions used to control a hardware device.

New types of sensors and actuators can be supported by adding them as a device library on the the `X10ABOT` motherboard. Creating a new device specification can be done without the need to understand any low level device specific coding, since all the configurations are abstracted for easier programming. Extensibility is therefore achieved through high-level abstraction of low level hardware components. The **X10** in `X10ABOT` which stands for extensibility, is one of the key design goals for the architecture. Extensibility defines the ability of the platform to support a vast variety of sensors and actuators with the inherent ability to support new or previously unsupported peripherals. This ability is a significant part of the core framework design. Both the hardware and software

allow for inclusion for new and cutting edge technologies by utilizing the simple idea of composition of low level microcode instructions. It can be shown that many seemingly complex protocols or devices operate using only a few simple fundamental electronic operations defined by our microcode instructions**CITE OR SHOW EXAMPLES** By creating a set of instructions that utilize these fundamental operations, any composite methods can be formed to complete most electronic tasks. For example: to operate a servo motor, this requires a variation in a pulse width modulated signal, a switch can be a simple digital I/O connection, and a light intensity sensor just simply uses an analogue to digital input. The `X10ABOT` system architecture was designed to facilitate device libraries which operate in a similar sense as hardware drivers for computer peripherals. A driver typically communicates with the device through a communications subsystem to which the hardware connects.

3.1.2.6 Multitasking

Another major aspect of the software system is its ability to execute simultaneous tasks. This was provided by the LEGO®Mindstorms®NXT platform and implemented in the X10ABOT robotics platform using the FreeRTOS real time scheduler. FreeRTOS is a tried and tested open source system which offers a very fast and efficient means of handling multiple tasks and timing operations.

3.1.2.7 The X10ABOT LEGO VM

There has been significant work toward the creation of a LEGO®Mindstorms®NXT virtual machine which will operate on top on the X10ABOT Library. User program code in the form of bytecode will be interpreted and executed with equivalent library commands. Many of the limitations in the hardware of the NXT are not enforced in its bytecode,

therefore it may have the ability to utilize the extra sensors and actuator ports provided by the X10ABOT Library.

3.2 The Arduino Implementation

`X10ABOT` is a robotics architecture that is indifferent to any particular hardware system. We endeavored to create an architecture that had very few hardware specific restrictions that would force the developer to chose one hardware platform over another. We also tried to create an architecture that was capable of running on systems as simple as 8-bit micro-controllers. What we found was that the majority of hobbyist robotics and robots used in school competitions tended to use 8-bit microcontrollers. However, many of the major notable robotics architectures were built on top of a full featured 32 or 64-bit processing systems**Elkady2012** The architecture should however be able to scale up to a full scale 32 or 64-bit computer. This will allow for more complex programs to be written with more processor and hardware intensive operations. The most basic requirements for any implementation of the `X10ABOT` architecture is a microprocessor which has the ability to perform digital and analogue input and output. In this case, one controller will operate as both motherboard and daughterboard. In order to support external daughterboards, each microcontroller must have support for the I^2C protocol in addition to the minimum requirements listed above.

The Arduino embedded system framework is a very active project, and it is notable for keeping at the cutting edge of embedded development. The framework is platform independent and currently deployed on a variety of processors including Atmel's AVR 8-bit and 32-bit ARM processors. There are even a few other companies including Microchip who have adopted the Arduino framework for their 32-bit processors as well**Development2012** The `X10ABOT`

software was written with the Processing language supported by Arduino which is a derivative of C++. This allowed us to take advantage of many of C++'s high level language features including class hierarchy which is the basis of how development for the motherboard libraries are done. Arduino also comes with a cross platform IDE that allows development on all the major operating systems.

The Arduino platform provides very inexpensive hardware that can be used for both motherboard and daughterboard. They were designed for physical add-ons called shields which are convenient connection boards that can host auxiliary circuitry like port headers, external power and H-bridges.

CHAPTER 4 RESULTS

This section seeks to demonstrate the steps taken in the data gathering process, critical data analysis and validation of the effectiveness of models developed. We start by looking at fluid flows in the heart of the UWICSS and extracting parameters for models in Section **??**. We can then demonstrate the effectiveness of the windkessel/lumped parameter model in predicting the behaviour of the aorta. In validating the hydraulic system for the UWICSS, we also examine the effect of the mechanical delays for the valves and pumps. These mechanical delays are a hindrance to the precise manipulation of fluid pressure and a direct limitation on the maximum frequency achievable by the controller.

This section also discusses the results drawn from the implementation of designs detailed in Chapter **??**. There are some important conclusions that can be made about the overall system and the numerical controller:

- The system based controller described in Chapter **??**, is capable of achieving a maximum pulse frequency of 0.6 pulse per second.
- The parameters of the two element windkessel model used in the system, can be successfully derived using known principles.
- The current firmware architecture can be used to implement the numerical controller discussed in Section **??** of this thesis.
- In order to achieve precise manipulation of fluid pressure between 0.4 pulse per second and a maximum of 6 pulses per second, faster actuations are required.

- Comparison of the performance results between the system with mechanical delays and that of the system without these delays, shows the reduction in the performance of the controller.

4.1 Results from Calibration of Sensors

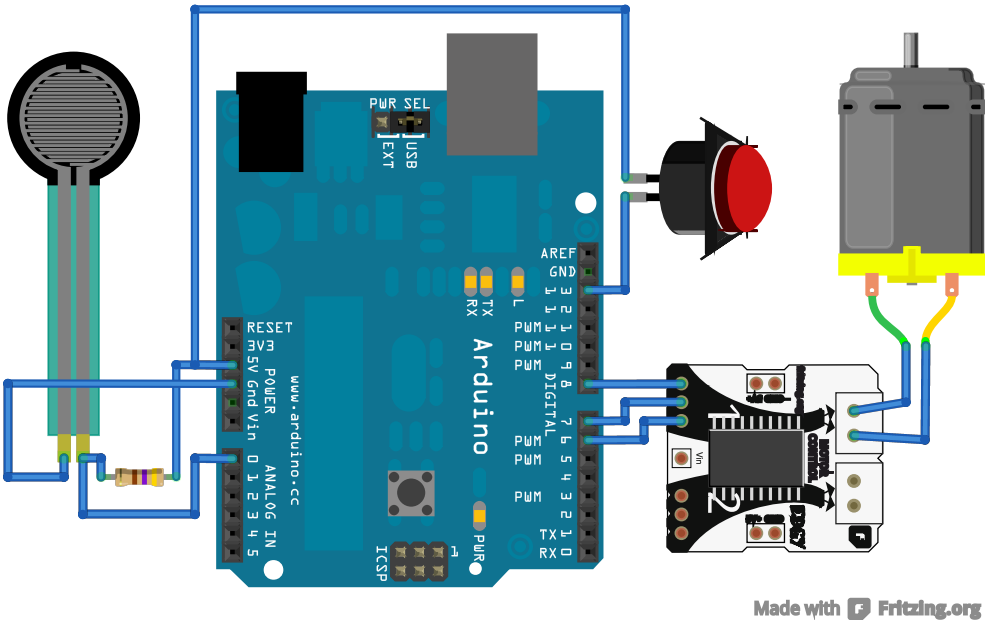

Figure 4.1. The components of a simple singleboard `X10ABOT` Example

The following chapter demonstrates how the `X10ABOT` architecture achieves its objectives through the hardware and software platform. We will demonstrate how the framework can be applied in a practical manner and we will explain the outcomes.

We will demonstrate the framework in action by showing excerpts of sample programs for a simple mobile robot with a number of sensors and actuators distributed across one or more daughterboards.

Code example 4 is a typical setup for a very simple mobile robot. All the sensors are on one daughterboard - in the case of our implementation, the “internal” daughterboard which is hosted on the same board as the motherboard. This robot has one actuator (a DC motor connected to an external H-bridge), a force sensor and a pushbutton sensor.

X10ABOT Sample code

```
/**
* Import the necessary libraries for the motherboard,
* daughterboard and the peripheral bus
**/
#include <Wire.h>
#include <X10ABOT_MB.h>
#include <X10ABOT_DB.h> //Included to support the internal daughterboard #0

//Declare and initialize motor on daughterboard #0, actuator port #1
Actuator motor1(0,1);
//Declare  and initialize force sensor on daughterboard #0, sensor port #1
Sensor force1(0,1);
//Declare  and initialize force sensor on daughterboard #0, sensor port #2
Sensor pushbutton(0,1,B);

void setup(){}

void loop(){
//Continuously check the sensors for a reading
if(pushbutton.readDigital() || (force1.readAnalogue()>100)){
  motor1.on(100); //Turn motor1 on at full power (100%)
  }else{
  motor1.off();  //Turn motor1 off
  }
}
```

Listing 4: Example of the X10ABOT architecture on a simple robot.

Code example 4 is a simple program that drives the motor in the forward(positive) direction at full speed if either the touch sensor records an input or if there is a force on the force sensor about a certain threshold, otherwise it will shut off the motors.

In this instance, there are two types of devices that are declared, **Sensors** and **Actuators**. These represent the parent classes of all sensors and actuators respectively in the `X10ABOT` architecture. All other sensor and actuator sub-types inherit from these two classes of devices, overriding where necessary.

It should be noted that it was necessary to include the library for both the motherboard and the daugterboard because the daughterboard and motherboard shares the same Arduino board. Daughterboard address #0 is reserved for the internal or resident daugterboard on the same physical Arduino platform.

4.2 Modularity

In the code example 5, we see how an extra sensor and an actuator was added for an extra functionality.

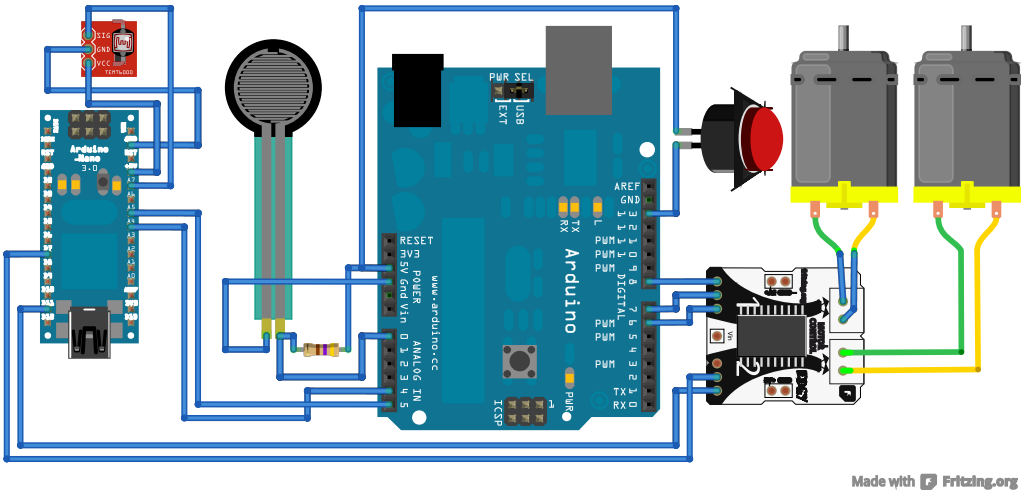

Figure 4.2. The components of a Modular singleboard `X10ABOT` Example

Modularity in Code

```
/**
* Import the necessary libraries for the motherboard,
* daughterboard and the peripheral bus
**/
#include <Wire.h>
#include <X10ABOT_MB.h>
#include <X10ABOT_DB.h> //Included to support the internal daughterboard #0

//Declare motor on daughterboard #0, actuator port #1
Actuator motor1(0,1);
//Declare force sensor on daughterboard #0, sensor port #1
Sensor force1(0,1);
//Declare force sensor on daughterboard #0, sensor port #2
Sensor pushbutton(0,1,B);
//Declare motor on daughterboard #9, actuator port #1
Actuator motor2(9,1);
//Declare force sensor on daughterboard #9, sensor port #1
Sensor lightSensor(9,5);

void setup(){}
void loop(){
//Continuously check the sensors for a reading
if(pushbutton.readDigital || force1.readAnalogue()){
  motor1.on(100); //Turn motor1 on at full power (100%)
}else{
  motor1.off();  //Turn motor1 off
  }
//Added extra functionality with a new sensor and a new actuator on daughterboard #9
//while motor1 is running and light is on the sensor, activate motor2
if (motor1.isOn() && lightSensor.readAnalogue())
  {motor2.on(-50); //Turn motor2 on at 50% in the reverse direction
  }else{
  motor2.off();  //Turn motor2 off
  }
}
```

Listing 5: Example of modularity in action.

In this particular example (code 5), even though *motor1* and *lightSensor*, two additional, independent devices were added to the system, it did not affect the existing code but fit seamlessly in the development process. Separate code was added to carry out its operation which never needed to interact with the exisiting system. There needed to be no special acccomodation for the new module in the existing system. There also was no hardware modification made to the existing setup even with the fact that an extra physical component, another daughterboard, was added to the system. All these features are an indication of a truly modular architecture. As previously defined, we included an entire subsystem without modifying the existing components. The modules can be removed just as easy as they were installed. The level of expertise required to modify this system is also relatively minimal due to its design.

4.3 Extensibility and Abstraction

We demonstrate the `X10ABOT` architecture's ability to support new and varied types of sensors and actuators. This is accomplished with no knowledge of the low-level details of the hardware. In the following example, we will define a simple, yet complete library that reads and interprets the data from a thermistor temperature sensor. The temperature value will be returned in fahrenheit units. This code was sourced from the Arduino Playground **therm** as a simple example to acquire thermistor temperature readings. We made a simple modification that allowed the same code to be applicable to the `X10ABOT` architecture.

Thermistor Library

```
#include <Sensor.h>
#include <math.h>

double readThermister() {
 int RawADC = analog(_db,_port);
 double Temp;
 Temp = log(((10240000/RawADC) - 10000));
 Temp = 1 / (0.001129148 + (0.000234125 + (0.0000000876741 * Temp * Temp ))* Temp );
 Temp = Temp - 273.15;            // Convert Kelvin to Celcius
 Temp = (Temp * 9.0)/ 5.0 + 32.0; // Convert Celcius to Fahrenheit
 return Temp;
}
```

Listing 6: Example: A complete library for a thermistor temperature sensor.

The library in code example 6 above is an almost exact replica of the fragment of code extracted from the Arduino library. The code accesses the analogue pin to read the sensor. The `X10ABOT` framework possesses special functions that access hardware components. For this particular case, the **analog(__db,__port)** function was invoked. This is a method used to abstract the analog pin so it becomes hardware independent. The analogue pin here is perceived as a virtual entity that belongs on a daughterboard **__db** on a particular port **__port**. The result is then returned to the calling function, presenting the temperature value to the user in Fahrenheit.

This thermistor library can then be easily utilized by the user as follows:

The code in code example 7 presents a temperature monitoring system that watches the temperature that it receives from two thermistors. If the value read from the thermistors goes beyond particular thresholds for either of them, an alarm is triggered.

Thermistor Example Application

```
#include <Wire.h>
#include <Thermistor.h>
#include <X10ABOT_MB.h>

//Declare motor on daughterboard #17, actuator port #1
Thermistor temp1(17,1);
//Declare force sensor on daughterboard #108, sensor port #6
Thermistor temp2(108,6);
//Declare digital alarm on daughterboard #10, sensor port #3
Buzzer alarm(10,3,A);

void setup(){}
void loop(){
//Continuously check the sensors to see if temperature within threshold
  if((temp1.readThermister() > 70 )|| temp1.readThermister()<30){
  alarm.on(); //Turn on alarm
}
```

Listing 7: Example application of the thermistor temperature sensor library.

The library for was included and it became pretty straightforward to manipulate the sensor afterwards.

4.4 Scalability

The results for the application on modularity are two fold and can be seen clearly to imply scalability. The method of modular additions is scalable to tens of modules as defined by the peripheral bus protocol. We therefore need to benchmark the performance of the architecture implemented on the Arduino platform. This is necessary to evaluate the speed at which instructions are excuted. This can determine how the system will perform when under duress

from multiple hardware devices simultaneously. These statistics can then be evaluated to deduce the architecture's suitability for a particular project. The X10ABOT architecture's major limiting factor is the speed of the peripheral bus, however, other hardware specific limitations may come into play. These may include, clock speed and available RAM, all of which may differ, even across Arduino hardware platforms. All tests below were carried out on the Arduino Mega, using the Atmel ™ATMega1280 microcontroller.

We carried out a few timing tests to evaluate the performance of the X10ABOT framework on the Arduino platform. The test were done using one motherboard and three daughterboards. The typical robotics project may never require as many daughter boards since one Arduino Mega may support up to eight sensor ports and six actuator ports. The smaller Arduino Uno can support up to three sensor ports and two actuator ports. Any combination of one or more of these Arduino boards can be used to implement the X10ABOT architecture. We did timing tests to measure the latency between the command request and the response. We also recorded the measurable difference when the peripheral bus is congested with a rapid succession of events for all the six daughterboards.

The tables below display the results of our timing experiments, the timing utility on the Arduino has a maximum resolution of 4 micro second intervals.

The latency for output operations was significantly less time than that of input operations. This was expected since there is no waiting period to request the value of the data from the port on the daughterboard for output operations.

Table 4.1: Table showing microcode latency in microseconds(μs) on the on-board daughteroard over 20 consecutive executions.

ONBOARD daughterboard

digitalOut	digitalIn	pwm	analogue
40	50084	60	50212
40	50088	60	50216
40	50084	60	50212
36	50084	60	50216
40	50084	56	50212
40	50084	60	50212
40	50084	60	50212
40	50080	60	50216
40	50084	60	50216
48	50080	56	50216
40	50088	56	50216
40	50084	56	50216
40	50084	60	50216
36	50084	60	50216
40	50080	60	50220
40	50084	64	50216
40	50084	56	50216
40	50080	60	50224
40	50084	56	50216
Average Times			
40.00	50083.58	58.95	50215.58

Table 4.2: Table showing microcode latency in microseconds(μs) on the off-board daughteroard over 20 consecutive executions.

OFFBOARD daughterboard

digitalOut	digitalIn	pwm	analogue
568	51320	688	51328
560	51320	684	51320
568	51324	684	51324
568	51324	684	51328
568	51332	684	51324
564	51324	684	51332
564	51324	684	51328
564	51320	688	51324
564	51320	680	51332
568	51324	688	51324
564	51324	680	51324

We carried out an experiment to determine how the system performed when executing a barrage of instruction and obtained the results lsted in table X below.

Table 4.3: Table showing microcode latency in microseconds(μs) on the off-board daughteroard for number of instructions executed(no.) and associated time it took to complete over 5050 consecutive executions of digitalOut.

OFFBOARD daughterboard barrage

no.	time (μs)	no.	time (μs)	no.	time (μs)	no.	time (μs)	no.	time (μs)
1	568	21	11840	41	23120	61	34380	81	45648
2	1132	22	12412	42	23676	62	34936	82	46216
3	1696	23	12972	43	24244	63	35516	83	46780
4	2268	24	13524	44	24796	64	36080	84	47352
5	2820	25	14108	45	25364	65	36620	85	47916
6	3384	26	14656	46	25932	66	37208	86	48480
7	3948	27	15228	47	26484	67	37772	87	49044
8	4520	28	15788	48	27068	68	38332	88	49600
9	5072	29	16352	49	27620	69	38880	89	50172
10	5636	30	16908	50	28188	70	39456	90	50736
11	6212	31	17472	51	28760	71	40028	91	51296
12	6768	32	18044	52	29316	72	40588	92	51860
13	7332	33	18600	53	29872	73	41156	93	52420
14	7896	34	19172	54	30452	74	41696	94	52980
15	8460	35	19732	55	31008	75	42276	95	53552
16	9020	36	20300	56	31552	76	42844	96	54112
17	9584	37	20856	57	32140	77	43408	97	54672
18	10148	38	21424	58	32696	78	43976	98	55244
19	10708	39	21988	59	33248	79	44536	99	55804
20	11280	40	22548	60	33828	80	45096	100	56364

Total time taken: 2847984

The results showed that even at high instruction thoroughput, the time to

execute a particular instruction does not vary significantly from its latency as a single instruction.

4.5 Economy

CHAPTER 5 CONCLUSION AND DISCUSSION

By actualizing the architectural design concepts of modularity, scalability and extensibility discussed in Chapter 2. We were able to create a platform that effectively accomplished these required objectives in a cost-effective manner. We have also realize that this design framework has created an extensible platform for numerous avenues of future work. There are opportunities to support various programming interfaces, porting `X10ABOT` to more powerful hardware and support of additional sensor and actuator devices.

5.1 Achievements

The following are a list of the main objectives which were accomplished:

- Designed modular, distributed hardware framework with pluggable peripheral boards
- Engineered a model that can support almost 900 sensors and 900 actuators
- Created an extensible software framework that can support the addition of many types of sensors & actuators
- Created a flexible system that allows easy interoperability with distinct hardware architectures
- Kept the cost low by using the Arduino as the main hardware component
- Successfully tested the framework with both digital and analog sensors and actuators for latency and scalability

5.2 Limitations

The efficiency of the X10ABOT architecture is limited by the speed of its peripheral bus, which in our case is I^2C and the power of the processors that implement it. The limitation details are listed below:

- Communication between motherboard and daughterboard are limited by the speed on the I^2C bus which is not only limited by its specification but the clock speed of the host processor.
- Our implementation was done on an 8-bit Arduino processor running at 16MHz with 8KB of SRAM. This inherently limits the kinds of tasks that could have been attempted.
- Other limitations include the I^2C 112 device limit on the peripheral bus, and the limit of 16 sensors and actuators (8 each) per daughterboard.

With the above listed discoveries, we can conclude that the X10ABOT architecture provides a suitable platform that will enable efficient robotics development. It will accomplish this by providing a model that can scale to a large number of sensors and actuators. This made possible by utilizing a distributed modular framework of daughterboards that can be attached whenever needed. We performed tests to measure the latency of I/O instructions between the peripheral devices and the motherboard. We found it to be in an acceptable range that was tolerable for most robotics applications within the scope. This delay time would also see improvements if the processor and communication speeds were increased. The performance of the system is dependent upon the hardware that implements it.

5.3 Future Work

In order to facilitate processor intensive operations, the architecture would have to be ported to a more powerful hardware platform. At the time of writing, there was a recent release of an Arduino based on the 32-bit, 84Mhz Arm core processor with 96 KBytes of SRAM. There is also the possibility of porting the architecture over to the Raspberry Pi for an even more powerful motherboard.

The code now has support for basic analogue and digital sensors. Future work could involve expanding the core offering for more advanced and popular sensors and actuators and the creation of an online library where a community could contribute libraries for their favorite sensors and actuators.

Currently the only supported method to write program code is using the Arduino framework. We had begun work on a bytecode interpreter that would allow the `X10ABOT` framework to read in bytecode from the LEGO Mindstorms' broad range of programming interfaces. Future work could see the completion of this interpreter that would provide a useful tool for roboticists with Mindstorms experience.

APPENDIX I - MODEL EQUATIONS